MODULE 10p - Contained Classes

The programs developed below illustrate various ways in which classes (and interfaces) may be be declared within other classes.

The first six examples are all jiffy programs which are intended to search through a static int array (whose identifier is data) and filter out, and print, any value which satisfies some acceptance criterion.

Each of the six programs incorporates an interface IntFilter which specifies that any implementing class should include a boolean method accept(). The accept() method in any implementing class is free to choose whatever acceptance criterion it likes but in these examples the satisfying criterion is simply 'any multiple of 5'.

Example A - ALL CLASSES AND INTERFACES AT THE TOP LEVEL

In the first example the method main() is in ClassExA and this class and both the interface IntFilter and another class Jack are at the top level.

```
public class ClassExA
 { private static int[] data = {11,13,15,17,19,20,23,25,27};

   public static void main(String[] args)
    { Jack j = new Jack();
      extract(j);
    }

   private static void extract(IntFilter infil)
    { for (int i=0; i<data.length; i++)
         if (infil.accept(data[i]))
            System.out.printf("%d%n", data[i]);
    }
 }

interface IntFilter
 { public abstract boolean accept(int n);
 }

class Jack implements IntFilter
 { public boolean accept(int n)
    { return n%5 == 0;
    }
 }
```

Key this in, compile it, and run it. The program works as follows:

1. A local variable j in method main() is assigned a new instance of Jack [a class which implements the interface IntFilter by supplying the method accept()].

2. Next j is handed to the extract() method whose formal argument is infil (and notice that j is of type Jack but infil is of the parent type IntFilter).

3. The extract() method picks out each value in the array data in turn and passes it to the infil.accept() method. It the result is true, the value is printed.

4. With the filter given, the values 15, 20 and 25 are printed.

This illustrates nothing new except the concept of a filter which in this case is rather a simple one. The interface IntFilter is an ordinary top-level interface and the class Jack is an ordinary top-level class.

Example B - NESTED TOP-LEVEL CLASS

The second example is almost exactly the same except that class Jack is declared nested within what is now ClassExB. It is declared both private and static, for much the same reasons that the date field and methods are (there will be more to say about this under Example C).

```
public class ClassExB
 { private static int[] data = {11,13,15,17,19,20,23,25,27};

   public static void main(String[] args)
    { Jack j = new Jack();                //  These two lines could be merged:
      extract(j);                         //        extract(new Jack());
    }

   private static void extract(IntFilter infil)
    { for (int i=0; i<data.length; i++)
         if (infil.accept(data[i]))
            System.out.printf("%d%n", data[i]);
    }

   private static class Jack implements IntFilter
    { public boolean accept(int n)
       { return n%5 == 0;
       }
    }
 }

interface IntFilter
 { public abstract boolean accept(int n);
 }
```

Key this in, compile it, and run it.

In this example Jack is called a 'nested top-level class'. It is nested within ClassExB but behaves exactly as it did in the previous program when it was at top level.

Java permits 'nested top-level interfaces' so IntFilter could have been incorporated into ClassExB as well as class Jack. There are though no other kinds of contained interface. By contrast there are several other ways in which classes may be contained within other classes...

INNER CLASSES

A nested top-level class is rather a special case of a contained class. It is really an ordinary top-level class that has been naively brought inside another class. The more interesting examples are known as 'inner classes' and there are three kinds:

1. Member classes
2. Local classes
3. Anonymous classes

Example C - MEMBER CLASS

In the second example class Jack had to be declared static. It is within ClassExB and if it were non-static it wouldn't ever be instantiated and so couldn't be used any more than the array data or the method extract() could be used if they were not declared static.

It is of course possible to declare the array data, the method extract() and the class Jack in a class of their own (ClExC in the program below) and instantiate this containing class from method main(). The instantiation makes all three usable. The following program illustrates this possibility.

```
public class ClassExC
 { public static void main(String[] args)
    { ClExC jill = new ClExC();            //  These two lines could be merged:
      jill.extract();                      //      (new ClExC()).extract();
    }
 }

class ClExC
 { private int[] data = {11,13,15,17,19,20,23,25,27};

   public void extract()
    { IntFilter infil = new Jack();
      for (int i=0; i<data.length; i++)
         if (infil.accept(data[i]))
            System.out.printf("%d%n", data[i]);
    }
```

```
   private class Jack implements IntFilter
    { public boolean accept(int n)
       { return n%5 == 0;
       }
    }
 }

interface IntFilter
 { public abstract boolean accept(int n);
 }
```

Key this in, compile it, and run it.

A class which is contained in the way that Jack is here is the first kind of inner class and is called a 'member class'. Its position within the containing class is as other class members, the data fields and methods.

Example D - LOCAL CLASS

A local class is positioned as a local variable, within a method. In the following program, class Jack is declared in method main() in just the same position as a local variable might be declared in the method.

```
public class ClassExD
 { private static int[] data = {11,13,15,17,19,20,23,25,27};

   public static void main(String[] args)
    { class Jack implements IntFilter
       { public boolean accept(int n)
          { return n%5 == 0;
          }
       }

      extract(new Jack());
    }

   private static void extract(IntFilter infil)
    { for (int i=0; i<data.length; i++)
         if (infil.accept(data[i]))
            System.out.printf("%d%n", data[i]);
    }
 }

interface IntFilter
 { public abstract boolean accept(int n);
 }
```

Key this in, compile it, and run it.

This is the second kind of inner class, a ‘local class’.

Example E - ANONYMOUS CLASS

The last kind of inner class is declared via the following syntax:

new <class-identifier> (<constructor arguments>) {<overriding methods>}

Recall that in ClassExC there was the declaration:

```
      IntFilter infil = new Jack();
```

in which class Jack, declared elsewhere, implemented the interface IntFilter. The idea of an anonymous class is to dispense with an identifier such as Jack and write instead:

```
      IntFilter infil = new IntFilter() {<blah-blah>}
```

where the methods which are required to implement IntFilter are supplied inside the curly brackets (the <blah-blah>!) rather than in a specific class such as Jack. The required overriding method in the present case is accept() and this is shown in the following example.

```
public class ClassExE
 { private static int[] data = {11,13,15,17,19,20,23,25,27};

   public static void main(String[] args)
    { IntFilter infil = new IntFilter(){public boolean accept(int n)
                                          { return n%5 == 0;
                                          }
                                       };
      extract(infil);
    }

   private static void extract(IntFilter infil)
    { for (int i=0; i<data.length; i++)
         if (infil.accept(data[i]))
            System.out.printf("%d%n", data[i]);
    }
 }

interface IntFilter
 { public abstract boolean accept(int n);
 }
```

Key this in, compile it, and run it.

This is the third kind of inner class, an ‘anonymous class’.

Example F - EVEN SLICKER ANONYMOUS CLASS

For those who want to write the absolute bare minimum there is no need
to write:

```
      IntFilter infil = <elaborate right-hand side>;

      extract(infil);
```

One can instead combine these two lines into:

```
      extract(<elaborate right-hand side>);
```

The following program illustrates this:

```
public class ClassExF
 { private static int[] data = {11,13,15,17,19,20,23,25,27};

   public static void main(String[] args)
    { extract(new IntFilter(){public boolean accept(int n)
                               { return n%5 == 0;
                               }
                             });
    }

   private static void extract(IntFilter infil)
    { for (int i=0; i<data.length; i++)
         if (infil.accept(data[i]))
            System.out.printf("%d%n", data[i]);
    }
 }

interface IntFilter
 { public abstract boolean accept(int n);
 }
```

Key this in, compile it, and run it.

This is not a new kind of inner class, simply an ‘anonymous class’
written in a way that takes some getting used to!

APPLICATION OF A NESTED TOP-LEVEL CLASS

Here is the PackTest program from way back rewritten so that class
Missive is a nested top-level class.

```
public class PackTest
 { public static void main(String[] args)
    { Missive ape = new Missive();
      System.out.printf("%S%n", ape.jill());
    }

   private static class Missive
    { public String jack()
       { return "JACK";
       }

      private String jill()
       { return "JILL";
       }
    }
 }
```

Key this in, compile it, and run it.

APPLICATION OF AN ANONYMOUS CLASS

Here is a the final version of the AppletC program rewritten so that
an anonymous class implements interface MouseListener instead of AppML
and another anonymous class implements interface MouseMotionListener
instead of AppMML.

```
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Polygon;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseMotionAdapter;
import java.awt.event.MouseEvent;

public class AppletC extends Applet
 { public Polygon poly = null;

   public void init()
    { this.addMouseListener
       (new MouseAdapter(){public void mousePressed(MouseEvent e)
                           { AppletC.this.poly = new Polygon();
                             AppletC.this.poly.addPoint(e.getX(), e.getY());
                             AppletC.this.repaint();
```

```
                            }
                          });

      this.addMouseMotionListener
       (new MouseMotionAdapter(){public void mouseDragged(MouseEvent e)
                                  { AppletC.this.poly.addPoint(e.getX(), e.getY());
                                    AppletC.this.repaint();
                                  }
                                 });
    }

   public void paint(Graphics g)
    { if (this.poly != null)
         g.drawPolygon(this.poly);
    }
 }
```

Key the above into the file AppletC.java and compile it.

Ensure that the following source is in the file AppletC.html

```
<HTML>
  <BODY>
    <APPLET code="AppletC.class" width=300 height=100>
            Java is not available.
    </APPLET>
  </BODY>
</HTML>
```

Give the following command:

```
$ appletviewer AppletC.html
```